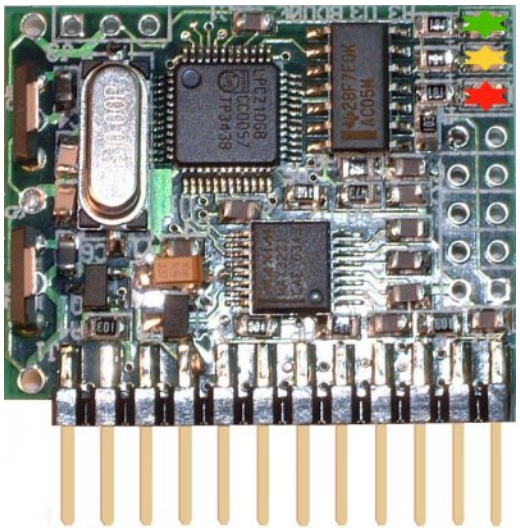
TiniARM Development Kit

User Manual V.3



NEW MICROS, INC.
1601 Chalk Hill Road
Dallas, Texas 75212
Tel: (214)-339-2204

Table of Contents

Compiling an Application for the TiniARM From Scratch

Programming the Arm processor in C can be done for free with the use of the freely available and multi-optioned GNU compilers. It can be a little daunting with 70MB downloads and thousands of pages of text to wade through to find the essential ingredients to make the recipe you're trying to implement. This manual assumes a neophyte level user and walks through the paces to get a bare application running on the TiniARM. With the knowledge of this journey, it is hoped that you will be able to forge your own.

NOTE: Changes from first manual (V.1) are marked with a change bar like the one on the left.

1.0 Introduction

The first job is to get programs and documentation. Some comes with the software but sometimes they need documentation. Don't get me wrong, it's great that there is a lot of documentation but it is vast and too many variables lie in the way of getting what I want. Where's my hello world program with an interrupt for only 234 bytes? Not 42K of behemoth layers of OS parts and services (which have their own empire of documentation). I often have to dig for a nugget of info to let me know which variable to modify each time something goes wrong in compiling, but to find that nugget again would be almost impossible. So here are the steps and nuggets (most of them) I took to get to the finish line.

To work with the target board you will need:

- the target board
- the development board
- power supply
- rs232 cable
- PC with COM port
- compiler
- downloader
- terminal program
- patience, luck and the NMI help forum

1.1 Overview

First, edit your source code; compile it with a compiler; link it and write it out with a loader; possibly translate the output to one suitable for downloading; download the result to the target board; attach test equipment and reset board to test it out.

Note all the steps and also note that this is a command line affair, no GUI. Although some are available, I have none to recommend yet. So crack the knuckles, rub the palms rapidly and push up the sleeves. In addition to being command line user interface (CLUI), it is Unixish and needs a Unix-like environment to run in. This means using the terminal window in Mac OSX and an extra download in Windows.

1.2 Get (with) the Program(s)

Whether you are running MacOSX, Windows or Linux, there's a precompiled application out there waiting to be downloaded. These are big and always come with many more things that you and all of your related family would never be able to use, so be prepared for a long download even on high speed connections. This document is tied to the gnude 1.2/1.1 release on Windows 2000 and MacOSX. Search for gnude download or if this link is valid:

http://sourceforge.net/projects/gnude/

Page down and download the appropriate version for your platform(s).

In addition, for the Windows platform you will also need to download from redhat:

http://www.redhat.com/download/cygwin.html

To communicate with the board you will need to run another program on the PC available from Philips at:

http://www.semiconductors.philips.com/files/products/standard/microcontrollers/utilities/lpc2000_flash_utility.zip

Download what you need and install (cygwin first on Windows).

A web page for the LPC2106 at Philips contains a useful number of links:

http://www.semiconductors.philips.com/pip/LPC2106.html

1.3 LPC2000 Flash Utility Notes

This tool is used to interact with the TiniARM processor over the serial port to move data between it, RAM, flash and the host computer. It is mainly used to program the processor. The following points will save some time working with this tool.

- The utility for communicating with the processor for transferring data to and from the chip cannot be used to create a file which is downloadable back to the chip because the first 0x3F bytes are not from the program in the rest of flash but are from the flash bootloader and when the contents of flash is transferred to the program buffer, it takes the first 0x3F bytes from the bootloader program and the rest of flash from whatever program is there. If this hex file is then put back into flash, the chip will not boot. To fix it, the first 0x3F bytes must come from a file source which has the values figured. Or the bytes can be edited in the buffer. The vector block has been shifted to 0x1E000 but the program prevents the user from seeing that part of flash.
- This has been fixed as of version V2.2.0 and is available from the LPC2000 files section in Yahoo groups: http://groups.yahoo.com/group/lpc2000/files/
- There is a button to calculate the security code at 0x14 but it is on the flash buffer tool and is not needed for normal downloads.
- The tool cannot handle files which are both RAM and Flash. You will get messages complaining about memory out of range. This could be something destined for RAM space like an initialized variable and could warrant a second look or it might not bother you at all.

1.4 Installing gnude on W2K

Download and install cygwin to create a unix environment under windows. Put gnude under "`/cygwin`". This is seen as just "`/`" inside a cygwin shell. To set the `PATH` variable:

```
export PATH=$PATH:/gnude/bin:/gnude/arm-elf/bin
```

This will give you access to the tools with or without their prefixes. This means if there is no other gcc tools installed in your paths, then gcc will be the same as `arm-elf-gcc` and can be used as a shortcut.

1.5 Included Files

- a.lst - assembly listing of example
- ivt.o - object file from compiling ivt.s
- ivt.s - source code containin the exception vectors for the beginning of flash
- libc.a - C library precompiled archive for use if needed
- lpc210x.h - header file with names for all the registers
- main.c - test program in C
- main.cmd - extra complicated loader script adapted from the tools
- main.hex - final output from compilation process
- main.map - map of symbols and memory
- main.o - compiled object from main.c
- main.out - the output from the loader/linker
- makefile - the script on how to make the test application
- simple.cmd - the script for where the test application fits in memory
- start.o - compiled object file for start.s
- start.s - start up code to set up the micro and transition into main in

2.0 Compiling

All the pieces for compiling in this example reside in one directory. This makes compiling simple. Just enter this command in the directory with everything in it:

```
make test
```

Like magic all the (and only the) required steps are executed to get from multiple source files and libraries to a singular hex file ready for download along with a memory map file and an assembled code listing. For the first run you should see something like this:

```
arm-elf-as   -o start.o start.s
arm-elf-as   -o ivt.o ivt.s
.compiling
..linking
arm-elf-ld -v -Map main.map -nostartfiles -T simple.cmd -o main.out  start.o ivt.o main.o libc.a
GNU ld version 2.14 20030612
...copying
arm-elf-objcopy -O ihex main.out main.hex
```

As projects go, they are seldom simple, once underway for any length of time, so it is best to have some organization up front in terms of structure. This is where the makefile and loader script come in. The makefile provides a singular point of controlling the compiling process and a way of managing compiler options and file groupings. The loader script is the recipe for all the ingredients from the compiler output and other precompiled archives (.a). The two files for these included here are simple and will need to be augmented as an application grows. Probably the next best thing to add to the makefile is some semblence of proper directory structure where source is kept separate from all the other pieces. The libc.a file is in the source directory for the simple directory structure in the makefile.

The following code example has been taken from application note 10254 and modified for this document and for the GNU compiler. But it is a working example which can be built upon. The interrupt does not work yet and has not been debugged other than a test to see if the interrupt routine was ever called (it wasn't), indicating probably a configuration problem.

2.1 Makefile

```
# for making arm code  Rob Chapman  Apr 1, 04

 NAME   = test io

CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy

.SUFFIXES : .o .c .s

CFLAGS  = -I./ -c -O3
AFLAGS  = -ahls -mapcs-32
CAFLAGS = $(CFLAGS) -Wa,-ahls,-mapcs-32
LFLAGS  =  -Map main.map -nostartfiles -T simple.cmd
CPFLAGS = -O ihex

test: main.out
    @ echo "...copying"
    $(CP) $(CPFLAGS) main.out main.hex

main.out: start.o ivt.o main.o simple.cmd
    @ echo "..linking"
    $(LD) $(LFLAGS) -o main.out  start.o ivt.o main.o libc.a

.c.o:
    @ echo ".compiling"
    @ $(CC) $(CAFLAGS) $<  > a.lst

mainin: start.s ivt.s main.c
```

```
    @ echo ".compiling"
    $(CC) $(CFLAGS) start.s ivt.s main.c
```

2.2 simple.cmd

```
/* Simple command script for organizing memory  Rob Chapman  Apr 1, 04 */

SECTIONS
{
    /* interrupt vectors */
    . = 0x0;  /* start of flash */
    .interp :
    {
    *(.interp)
    }

    /* code and constants */
    .text :
    {
    *(.text)
    *(.strings)
    *(.rodata.*)
    *(.init)
    *(.comment)
    }

    /* uninitialized data */
    . = 0x40000000;  /* start of ram */
    .bss :
    {
    *(.bss)
    *(COMMON)
    *(.data)
    }
}
```

2.3 main.c

```
/* code originally from phillips appnote 10254 */

/* ******************************************************
               Function declarations
  ****************************************************** */
void IRQHandler(void);
void feed(void);
void Initialize(void);

/******************************************************
                  Header files
 ******************************************************/
#include "LPC210x.h"
#include <string.h>

// serial port
#define RDR  0x01
```

```
#define THRE 0x20

char rx_query(void)
{
          return UART0_LSR & RDR;
}

char tx_query(void)
{
          return UART0_LSR & THRE;
}

void tx(char c)
{
          UART0_THR = c;
}

char rx(void)
{
          return UART0_RBR;
}

void tx_str(char *s)  // send a string
{
          while(*s)
                        if (tx_query())
                                      tx(*s++);
}

/*******************************************************
                     MAIN
*******************************************************/

void __main (void)
{
          char *x = (char *)0, *y = (char *)0x1000;

  /* Initialize the system */
  Initialize();

  /* Start timer */
// Interrupts not working yet so this code is commented out for now
//  TIMER1_TCR=0x1;

// For testing library inclusion
//  memcpy(x,y,100);

  // test banner
  tx_str("\n\rTest echo\n\r");
  // echo for main loop
  while(1)
          if (rx_query() && tx_query())  // received and ready to send
                        tx(rx());  // get and send
}
```

```
/*********************************************************
                      Initialize
*********************************************************/
extern const char _text_start, _text_end;
extern char _data_start, _data_end;
#define PLOCK 0x400

void Initialize(void)
{
//  memcpy(&_data_start, &_text_end, &_data_end - &_data_start);

  // set io pins for leds red off, yellow off, green on
  IODIR |= 0x03800000;  // 23-25 are outputs
  IOSET = 0x00800000;   // green led on
  IOCLR = 0x03000000;   // red and yellow off

  /*
   * Initialize PLL (Configured for a 10MHz crystal) to
   * boost processor clock to 60MHz
   */

  /* Setting Multiplier and divider values */
  PLLCFG=0x25;
  feed();

  /* Enabling the PLL */
  PLLCON=0x1;
  feed();

  /* Wait for the PLL to lock to set frequency */
  while(!(PLLSTAT & PLOCK)){}

  /* Connect the PLL as the clock source */
  PLLCON=0x3;
  feed();

 /*
  * Enabling MAM and setting number of clocks used for
  * Flash memory fetch
  */
  MAMCR=0;  // make sure MAM is off before adjusting
  MAMTIM=0x3;
  MAMCR=0x2;

 /*
  * Setting peripheral Clock (pclk) to System
  * Clock (cclk)
  */
  VPBDIV=0x1;

  /* Initialize GPIO */
//  IODIR=0xFFFF;
//  IOSET=0xFFFF;

  /* Initialize Timer 1 */
```

```
  TIMER1_TCR=0x0;
  TIMER1_TC=0x0;
  TIMER1_PR=0x0;
  TIMER1_PC=0x0;

  /* End user has to fill in the match value */
  TIMER1_MR0=0x123456;

  /* Reset and interrupt on match */
  TIMER1_MCR=0x3;

  /* Initialize VIC */
  VICIntSelect=0x0; /* Timer 1 selected as IRQ */
  VICIntEnable= 0x20; /* Timer 1 interrupt enabled */
  VICVectCntl0= 0x25;

  /* Address of the ISR */
  VICVectAddr0=(unsigned long)IRQHandler;


  /* initialize serial port */
  // initialize UART
  PINSEL0 = 5;      // enable UART0 in/out
  UART0_FCR = 0x7;   // enable and reset fifos
  UART0_LCR = 0x83;// 8 bits; enable divisor latches
  UART0_DLL = 0x87;  // LSB divider for 60mhz to be 9600x16
  UART0_DLM = 0x01;  // MSB
  UART0_LCR = 0x3;   // disable divisor latches
}

/*********************************************************
                       Timer 1 ISR
 *********************************************************/
void __attribute__((interrupt)) IRQHandler(void)
{
/*
 * The Interrupt Service Routine code will come here. The
 * interrupt needs to be cleared in Timer1 and a write must
 * be performed on the VIC Vector Address Register to
 * update the VIC priority hardware. Here the user could
 * blink a few LED's or toggle some port pins as an
 * indication of being in the ISR
 */
  IOSET = 0x01000000;   // yellow led on
  TIMER1_IR=0x1;
  VICVectAddr=0xff;
}

void feed(void)
{
  PLLFEED=0xAA;
  PLLFEED=0x55;
}
```

2.4 ivt.s

```
@  code originally from phillips appnote 10254
@ ----------------------------------------------------------
@               Assembler Directives
@ ----------------------------------------------------------
               .section .interp,"ax"  @ alocateable and executable New Code section
                               @ compiler option : CODE32          @ ARM code
               .extern start   @ start symbol not
                               @ defined in this
                               @ section
Entry:                          @ Defines entry point
@ ----------------------------------------------------------
               LDR PC, =_start
               LDR PC, Undefined_Addr
               LDR PC, SWI_Addr
               LDR PC, Prefetch_Addr
               LDR PC, Abort_Addr

@ At 0x14 the user should insert a signature (checksum).
@ This signature enables the bootloader to determine if
@ there is valid user code in the Flash. Currently most of
@ the Flash programming tools (debuggers and ISP utility)
@ have this feature built–in so the end user need not worry
@ about it. If the tool does not provide this feature then
@ the value has to be computed manually and has to be
@ inserted at 0x14. Details on computation of checksum
@ could be found in the Flash programming chapter in the
@ LPC2104/5/6 User Manual.

               NOP                       @ for code to be
               LDR PC, [PC, #0xFFFFF010] @ load irq vector from vic
               LDR PC, FIQ_Addr
Undefined_Addr: .word Undefined_Handler
SWI_Addr:       .word SWI_Handler
Prefetch_Addr:  .word Prefetch_Handler
Abort_Addr:     .word Abort_Handler
FIQ_Addr:       .word FIQ_Handler

@ ----------------------------------------------------------
@ Exception Handlers
@ ----------------------------------------------------------
@ The following dummy handlers do not do anything useful in
@ this example. They are set up here for completeness.

Undefined_Handler:
               B         Undefined_Handler
SWI_Handler:
               B         SWI_Handler
Prefetch_Handler:
               B         Prefetch_Handler
Abort_Handler:
               B         Abort_Handler
FIQ_Handler:
               B         FIQ_Handler
```

```
END:
```

2.5 start.s

```
@// code originally from phillips appnote 10254
@ ----------------------------------------------------------
@                       Assembler Directives
@ ----------------------------------------------------------
               .section asm_code,"ax"   @ New Code section
               @ CODE32                 @ ARM code
               .extern __main          @ main not defined
                                       @ in this section
               .global _start           @ global symbol
                                       @ referenced in
                                       @ ivt.s
@ ----------------------------------------------------------
_start:
               @ Set SP for Supervisor mode. Depending upon
               @ the stack the application needs this value
               @ needs to be set.
               @ stack is already set by bootloader
               @ but if this point is entered by any
               @ other means than reset, the stack pointer
               @ needs to be set explicity

               @ LDR SP,=0x40001000

               @ Setting up SP for IRQ and FIQ mode.
               @ Change mode before setting each one
               @ move back again to Supervisor mode
               @ Each interrupt has its own link
               @ register, stack pointer and program
               @ counter The stack pointers must be
               @ initialized for interrupts to be
               @ used later.

               @ setup for fiq and irq interrupt stacks to run
               @ below current stack by 1000.
               mov r0, sp         @ copy current stack pointer
               sub r0, r0, #1000  @ make irq stack pointer
               sub r1, r0, #1000  @ make fiq stack pointer
               msr cpsr_c, #0x12  @ switch to irq mode
               mov sp, r0         @ set irq stack pointer
               msr cpsr_c, #0x11  @ fiq mode
               mov sp, r1         @ set fiq stack pointer
               msr cpsr_c, #0x13  @ supervisor mode F,I enabled

               @ Jump to C code

               LDR lr, =__main
               MOV pc, lr
```

3.0 Downloading

Once you've produced the Intel .hex file output as the final result of compiling, linking and translating, it is time to download it to the

component board. A word of note here to the unwary of jargon. Sometimes you will see upload to flash other times you will see download to flash. As either transaction is just electrons jiggling in a wire, the direction doesn't make sense, so just pay attention to the destination.

The flash utility is fairly straight forward. Press the ... button to set the flash programming filename to the .hex file generated after all the making is done (main.hex). Then press the update flash button. The first time is likely to fail because you've run your test program. So wait for tthe timeout and error dialog and try again. This time you will receive a dialog asking you to reset the board. Make sure the reboot jumper is on before you press the reset button. After you press the reset button, then click on the dialog and in about 10 seconds, the download should be finished and it can be tested.

4.0 Testing

This phase usually ends quickly. The biggest part is the setup.

Switch either cables plugged into the board for a different com port hookup or switch comports between the download application and the terminal application unless an LED display is fine for your purposes (With three colors, 8 states are possible.). The other thing to remember is to remove the reboot jumper. Otherwise you will just boot back into the boot software. Press the reset pin with your terminal program set to 9600, 8 bits and no parity. Hyperterm on Windows or Zterm on Mac OSX.

5.0 Final Result

There is a lot to get right and when it does you should see:

```
Test echo
```

and if you type, it should be echoed. Congratulations

6.0 Helps

These are the help screens from some of the tools and can be a useful reference:

6.1 Compiler: arm-elf-gcc --help

```
Usage: arm-elf-gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path
  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
```

```
  -V <version>              Run gcc version number <version>, if installed
  -v                        Display the programs invoked by the compiler
  -###                      Like -v but options quoted and commands not executed
  -E                        Preprocess only; do not compile, assemble or link
  -S                        Compile only; do not assemble or link
  -c                        Compile and assemble, but do not link
  -o <file>                 Place the output into <file>
  -x <language>             Specify the language of the following input files
                            Permissible languages include: c c++ assembler none
                            'none' means revert to the default behavior of
                            guessing the language based on the file's extension

Options starting with -g, -f, -m, -O, -W, or --param are automatically
 passed on to the various sub-processes invoked by arm-elf-gcc.  In order to pass
 other options on to these processes the -W<letter> options must be used.
```

6.1.1 Compiler Target: arm-elf-gcc --target-help

```
Target specific options:
  -mcaller-super-interworki Thumb: Assume function pointers may go to non-Thumb aware code
  -mcallee-super-interworki Thumb: Assume non-static functions may be called from ARM code
  -mtpcs-leaf-frame         Thumb: Generate (leaf) stack frames even if not needed
  -mtpcs-frame              Thumb: Generate (non-leaf) stack frames even if not needed
  -mthumb                   Compile for the Thumb not the ARM
  -mlong-calls              Generate call insns as indirect calls, if necessary
  -msingle-pic-base         Do not load the PIC register in function prologues
  -mno-sched-prolog         Do not move instructions into a function's prologue
  -mabort-on-noreturn       Generate a call to abort if a noreturn function returns
  -mthumb-interwork         Support calls between Thumb and ARM instruction sets
  -mwords-little-endian     Assume big endian bytes, little endian words
  -mlittle-endian           Assume target CPU is configured as little endian
  -mbig-endian              Assume target CPU is configured as big endian
  -mhard-float              Use hardware floating point instructions
  -msoft-float              Use library calls to perform FP operations
  -malignment-traps         The MMU will trap on unaligned accesses
  -mapcs-reentrant          Generate re-entrant, PIC code
  -mapcs-float              Pass FP arguments in FP registers
  -mapcs-26                 Use the 26-bit version of the APCS
  -mapcs-32                 Use the 32-bit version of the APCS
  -mpoke-function-name      Store function names in object code
  -mapcs-frame              Generate APCS conformant stack frames
  -mpic-register=           Specify the register to be used for PIC addressing
  -mstructure-size-boundary Specify the minimum bit alignment of structures
  -mfp=                     Specify the version of the floating point emulator
  -march=                   Specify the name of the target architecture
  -mcpu=                    Specify the name of the target CPU
 ARM-specific assembler options:
  -k                       generate PIC code
  -mthumb                  assemble Thumb code
  -mthumb-interwork        support ARM/Thumb interworking
  -moabi                   use old ABI (ELF only)
  -mapcs-32                code uses 32-bit program counter
  -mapcs-26                code uses 26-bit program counter
  -mapcs-float             floating point args are in fp regs
  -mapcs-reentrant         re-entrant code
```

```
  -matpcs                 code is ATPCS conformant
  -mbig-endian            assemble for big-endian
  -mlittle-endian         assemble for little-endian
  -mapcs-frame            use frame pointer
  -mapcs-stack-check      use stack size checking
  -mcpu=<cpu name>        assemble for CPU <cpu name>
  -march=<arch name>      assemble for architecture <arch name>
  -mfpu=<fpu name>        assemble for FPU architecture <fpu name>
  -EB                     assemble code for a big-endian cpu
  -EL                     assemble code for a little-endian cpu
armelf:
  -Bgroup               Selects group name lookup rules for DSO
  --disable-new-dtags   Disable new dynamic tags
  --enable-new-dtags    Enable new dynamic tags
  --eh-frame-hdr        Create .eh_frame_hdr section
  -z combreloc          Merge dynamic relocs into one section and sort
  -z defs               Disallows undefined symbols
  -z initfirst          Mark DSO to be initialized first at runtime
  -z interpose          Mark object to interpose all DSOs but executable
  -z loadfltr           Mark object requiring immediate process
  -z muldefs            Allow multiple definitions
  -z nocombreloc        Don't merge dynamic relocs into one section
  -z nocopyreloc        Don't create copy relocs
  -z nodefaultlib       Mark object not to use default search paths
  -z nodelete           Mark DSO non-deletable at runtime
  -z nodlopen           Mark DSO not available to dlopen
  -z nodump             Mark DSO not available to dldump
  -z now                Mark object non-lazy runtime binding
  -z origin             Mark object requiring immediate $ORIGIN processing
                          at runtime
  -z KEYWORD            Ignored for Solaris compatibility
  -p --no-pipeline-knowledge  Stop the linker knowing about the pipeline length
     --thumb-entry=<sym>      Set the entry point to be Thumb symbol <sym>
```

6.2 Assembler: arm-elf-as --help

```
Usage: arm-elf-as [option...] [asmfile...]
Options:
  -a[sub-option...]       turn on listings
                          Sub-options [default hls]:
                          c      omit false conditionals
                          d      omit debugging directives
                          h      include high-level source
                          l      include assembly
                          m      include macro expansions
                          n      omit forms processing
                          s      include symbols
                          =FILE  list to FILE (must be last sub-option)
  -D                      produce assembler debugging messages
  --defsym SYM=VAL        define symbol SYM to given value
  -f                      skip whitespace and comment preprocessing
  --gstabs                generate stabs debugging information
  --gdwarf2               generate DWARF2 debugging information
  --help                  show this message and exit
  --target-help           show target specific options
```

```
  -I DIR                  add DIR to search list for .include directives
  -J                      don't warn about signed overflow
  -K                      warn when differences altered for long displacements
  -L,--keep-locals        keep local symbols (e.g. starting with `L')
  -M,--mri                assemble in MRI compatibility mode
  --MD FILE               write dependency information in FILE (default none)
  -nocpp                  ignored
  -o OBJFILE              name the object-file output OBJFILE (default a.out)
  -R                      fold data section into text section
  --statistics            print various measured statistics from execution
  --strip-local-absolute  strip local absolute symbols
  --traditional-format    Use same format as native assembler when possible
  --version               print assembler version number and exit
  -W  --no-warn           suppress warnings
  --warn                  don't suppress warnings
  --fatal-warnings        treat warnings as errors
  --itbl INSTTBL          extend instruction set to include instructions
                          matching the specifications defined in file INSTTBL
  -w                      ignored
  -X                      ignored
  -Z                      generate object file even after errors
  --listing-lhs-width     set the width in words of the output data column of
                          the listing
  --listing-lhs-width2    set the width in words of the continuation lines
                          of the output data column; ignored if smaller than
                          the width of the first line
  --listing-rhs-width     set the max width in characters of the lines from
                          the source file
  --listing-cont-lines    set the maximum number of continuation lines used
                          for the output data column of the listing
ARM-specific assembler options:
  -k                      generate PIC code
  -mthumb                 assemble Thumb code
  -mthumb-interwork       support ARM/Thumb interworking
  -moabi                  use old ABI (ELF only)
  -mapcs-32               code uses 32-bit program counter
  -mapcs-26               code uses 26-bit program counter
  -mapcs-float            floating point args are in fp regs
  -mapcs-reentrant        re-entrant code
  -matpcs                 code is ATPCS conformant
  -mbig-endian            assemble for big-endian
  -mlittle-endian         assemble for little-endian
  -mapcs-frame            use frame pointer
  -mapcs-stack-check      use stack size checking
  -mcpu=<cpu name>        assemble for CPU <cpu name>
  -march=<arch name>      assemble for architecture <arch name>
  -mfpu=<fpu name>        assemble for FPU architecture <fpu name>
  -EB                     assemble code for a big-endian cpu
  -EL                     assemble code for a little-endian cpu
```

6.3 Loader: arm-elf-ld --help

```
Usage: arm-elf-ld [options] file...
Options:
  -a KEYWORD                  Shared library control for HP/UX compatibility
```

```
-A ARCH, --architecture ARCH
                           Set architecture
-b TARGET, --format TARGET  Specify target for following input files
-c FILE, --mri-script FILE  Read MRI format linker script
-d, -dc, -dp                Force common symbols to be defined
-e ADDRESS, --entry ADDRESS Set start address
-E, --export-dynamic        Export all dynamic symbols
-EB                         Link big-endian objects
-EL                         Link little-endian objects
-f SHLIB, --auxiliary SHLIB Auxiliary filter for shared object symbol table
-F SHLIB, --filter SHLIB    Filter for shared object symbol table
-g                          Ignored
-G SIZE, --gpsize SIZE      Small data size (if no size, same as --shared)
-h FILENAME, -soname FILENAME
                           Set internal name of shared library
-I PROGRAM, --dynamic-linker PROGRAM
                           Set PROGRAM as the dynamic linker to use
-l LIBNAME, --library LIBNAME
                           Search for library LIBNAME
-L DIRECTORY, --library-path DIRECTORY
                           Add DIRECTORY to library search path
-m EMULATION                Set emulation
-M, --print-map             Print map file on standard output
-n, --nmagic                Do not page align data
-N, --omagic                Do not page align data, do not make text readonly
--no-omagic                 Page align data, make text readonly
-o FILE, --output FILE      Set output file name
-O                          Optimize output file
-Qy                         Ignored for SVR4 compatibility
-q, --emit-relocs           Generate relocations in final output
-r, -i, --relocateable      Generate relocateable output
-R FILE, --just-symbols FILE
                           Just link symbols (if directory, same as --rpath)
-s, --strip-all             Strip all symbols
-S, --strip-debug           Strip debugging symbols
--strip-discarded           Strip symbols in discarded sections
--no-strip-discarded        Do not strip symbols in discarded sections
-t, --trace                 Trace file opens
-T FILE, --script FILE      Read linker script
-u SYMBOL, --undefined SYMBOL
                           Start with undefined reference to SYMBOL
--unique [=SECTION]         Don't merge input [SECTION | orphan] sections
-Ur                         Build global constructor/destructor tables
-v, --version               Print version information
-V                          Print version and emulation information
-x, --discard-all           Discard all local symbols
-X, --discard-locals        Discard temporary local symbols (default)
--discard-none              Don't discard any local symbols
-y SYMBOL, --trace-symbol SYMBOL
                           Trace mentions of SYMBOL
-Y PATH                     Default search path for Solaris compatibility
-(, --start-group           Start a group
-), --end-group             End a group
--accept-unknown-input-arch Accept input files whose architecture cannot be determined
--no-accept-unknown-input-arch
```

```
                          Reject input files whose architecture is unknown
-assert KEYWORD           Ignored for SunOS compatibility
-Bdynamic, -dy, -call_shared
                          Link against shared libraries
-Bstatic, -dn, -non_shared, -static
                          Do not link against shared libraries
-Bsymbolic                Bind global references locally
--check-sections          Check section addresses for overlaps (default)
--no-check-sections       Do not check section addresses for overlaps
--cref                    Output cross reference table
--defsym SYMBOL=EXPRESSION  Define a symbol
--demangle [=STYLE]       Demangle symbol names [using STYLE]
--embedded-relocs         Generate embedded relocs
-fini SYMBOL              Call SYMBOL at unload-time
--force-exe-suffix        Force generation of file with .exe suffix
--gc-sections             Remove unused sections (on some targets)
--no-gc-sections          Don't remove unused sections (default)
--help                    Print option help
-init SYMBOL              Call SYMBOL at load-time
-Map FILE                 Write a map file
--no-define-common        Do not define Common storage
--no-demangle             Do not demangle symbol names
--no-keep-memory          Use less memory and more disk I/O
--no-undefined            Allow no undefined symbols
--allow-shlib-undefined   Allow undefined symbols in shared objects (the default)
--no-allow-shlib-undefined  Do not allow undefined symbols in shared objects
--allow-multiple-definition Allow multiple definitions
--no-undefined-version    Disallow undefined version
--no-warn-mismatch        Don't warn about mismatched input files
--no-whole-archive        Turn off --whole-archive
--noinhibit-exec          Create an output file even if errors occur
-nostdlib                 Only use library directories specified on
                            the command line
--oformat TARGET          Specify target of output file
-qmagic                   Ignored for Linux compatibility
--relax                   Relax branches on certain targets
--retain-symbols-file FILE  Keep only symbols listed in FILE
-rpath PATH               Set runtime shared library search path
-rpath-link PATH          Set link time shared library search path
-shared, -Bshareable      Create a shared library
--sort-common             Sort common symbols by size
--spare-dynamic-tags COUNT  How many tags to reserve in .dynamic section
--split-by-file [=SIZE]   Split output sections every SIZE octets
--split-by-reloc [=COUNT]   Split output sections every COUNT relocs
--stats                   Print memory usage statistics
--target-help             Display target specific options
--task-link SYMBOL        Do task level linking
--traditional-format      Use same format as native linker
--section-start SECTION=ADDRESS
                          Set address of named section
-Tbss ADDRESS             Set address of .bss section
-Tdata ADDRESS            Set address of .data section
-Ttext ADDRESS            Set address of .text section
--verbose                 Output lots of information during link
--version-script FILE     Read version information script
```

```
  --version-exports-section SYMBOL
                              Take export symbols list from .exports, using
                                SYMBOL as the version.
  --warn-common               Warn about duplicate common symbols
  --warn-constructors         Warn if global constructors/destructors are seen
  --warn-multiple-gp          Warn if the multiple GP values are used
  --warn-once                 Warn only once per undefined symbol
  --warn-section-align        Warn if start of section changes due to alignment
  --fatal-warnings            Treat warnings as errors
  --whole-archive             Include all objects from following archives
  --wrap SYMBOL               Use wrapper functions for SYMBOL
  --mpc860c0 [=WORDS]         Modify problematic branches in last WORDS (1-10,
                                default 5) words of a page
arm-elf-ld: supported targets: elf32-littlearm elf32-bigarm elf32-little elf32-big
                               srec symbolsrec tekhex binary ihex
arm-elf-ld: supported emulations: armelf
arm-elf-ld: emulation specific options:
armelf:
  -Bgroup               Selects group name lookup rules for DSO
  --disable-new-dtags   Disable new dynamic tags
  --enable-new-dtags    Enable new dynamic tags
  --eh-frame-hdr        Create .eh_frame_hdr section
  -z combreloc          Merge dynamic relocs into one section and sort
  -z defs               Disallows undefined symbols
  -z initfirst          Mark DSO to be initialized first at runtime
  -z interpose          Mark object to interpose all DSOs but executable
  -z loadfltr           Mark object requiring immediate process
  -z muldefs            Allow multiple definitions
  -z nocombreloc        Don't merge dynamic relocs into one section
  -z nocopyreloc        Don't create copy relocs
  -z nodefaultlib       Mark object not to use default search paths
  -z nodelete           Mark DSO non-deletable at runtime
  -z nodlopen           Mark DSO not available to dlopen
  -z nodump             Mark DSO not available to dldump
  -z now                Mark object non-lazy runtime binding
  -z origin             Mark object requiring immediate $ORIGIN processing
                          at runtime
  -z KEYWORD            Ignored for Solaris compatibility
  -p --no-pipeline-knowledge  Stop the linker knowing about the pipeline length
     --thumb-entry=<sym>      Set the entry point to be Thumb symbol <sym>
```

6.4 Translation: arm-elf-objcopy --help

```
Usage: arm-elf-objcopy [option(s)] in-file [out-file]
 Copies a binary file, possibly transforming it in the process
 The options are:
  -I --input-target <bfdname>      Assume input file is in format <bfdname>
  -O --output-target <bfdname>     Create an output file in format <bfdname>
  -B --binary-architecture <arch>  Set arch of output file, when input is binary
  -F --target <bfdname>            Set both input and output format to <bfdname>
     --debugging                   Convert debugging information, if possible
  -p --preserve-dates              Copy modified/access timestamps to the output
  -j --only-section <name>         Only copy section <name> into the output
  -R --remove-section <name>       Remove section <name> from the output
  -S --strip-all                   Remove all symbol and relocation information
```

```
  -g --strip-debug                Remove all debugging symbols
     --strip-unneeded             Remove all symbols not needed by relocations
  -N --strip-symbol <name>        Do not copy symbol <name>
  -K --keep-symbol <name>         Only copy symbol <name>
  -L --localize-symbol <name>     Force symbol <name> to be marked as a local
  -G --keep-global-symbol <name>  Localize all symbols except <name>
  -W --weaken-symbol <name>       Force symbol <name> to be marked as a weak
     --weaken                     Force all global symbols to be marked as weak
  -x --discard-all                Remove all non-global symbols
  -X --discard-locals             Remove any compiler-generated symbols
  -i --interleave <number>        Only copy one out of every <number> bytes
  -b --byte <num>                 Select byte <num> in every interleaved block
     --gap-fill <val>             Fill gaps between sections with <val>
     --pad-to <addr>              Pad the last section up to address <addr>
     --set-start <addr>           Set the start address to <addr>
    {--change-start|--adjust-start} <incr>
                                  Add <incr> to the start address
    {--change-addresses|--adjust-vma} <incr>
                                  Add <incr> to LMA, VMA and start addresses
    {--change-section-address|--adjust-section-vma} <name>{=|+|-}<val>
                                  Change LMA and VMA of section <name> by <val>
     --change-section-lma <name>{=|+|-}<val>
                                  Change the LMA of section <name> by <val>
     --change-section-vma <name>{=|+|-}<val>
                                  Change the VMA of section <name> by <val>
    {--[no-]change-warnings|--[no-]adjust-warnings}
                                  Warn if a named section does not exist
     --set-section-flags <name>=<flags>
                                  Set section <name>'s properties to <flags>
     --add-section <name>=<file>  Add section <name> found in <file> to output
     --rename-section <old>=<new>[,<flags>] Rename section <old> to <new>
     --change-leading-char        Force output format's leading character style
     --remove-leading-char        Remove leading character from global symbols
     --redefine-sym <old>=<new>   Redefine symbol name <old> to <new>
     --srec-len <number>          Restrict the length of generated Srecords
     --srec-forceS3               Restrict the type of generated Srecords to S3
     --strip-symbols <file>       -N for all symbols listed in <file>
     --keep-symbols <file>        -K for all symbols listed in <file>
     --localize-symbols <file>    -L for all symbols listed in <file>
     --keep-global-symbols <file> -G for all symbols listed in <file>
     --weaken-symbols <file>      -W for all symbols listed in <file>
     --alt-machine-code <index>   Use alternate machine code for output
     --prefix-symbols <prefix>    Add <prefix> to start of every symbol name
     --prefix-sections <prefix>   Add <prefix> to start of every section name
     --prefix-alloc-sections <prefix>
                                  Add <prefix> to start of every allocatable
                                    section name
  -v --verbose                    List all object files modified
  -V --version                    Display this program's version number
  -h --help                       Display this output
     --info                       List object formats & architectures supported
arm-elf-objcopy: supported targets: elf32-littlearm elf32-bigarm elf2-little elf32-big
                                  srec symbolsrec tekhex binary ihex
```